RWM\_1718\_P3

Component: Level Loading

C00195564: Daragh Behan

# User Guide

[**User Guide**](#_qk5uz8bbycvr) **1**

[Description of this Component](#_kzvp76o6jicg) 1

[How to include this into your project](#_rwx2pnagfzmq) 1

[The important Bits](#_ix2r2pi2w2gs) 2

[Setting up the .json Loader in your project](#_cp59oul1gtc8) 2

[Example: .JSON file creation](#_1vdvv7o4mw6q) 2

[Example: Loading general objects](#_aw599ptgpnl8) 4

[Loading Box2D Bodies](#_iizpafe4irp5) 5

[Features](#_n2l28z9u85ym) 6

[Loading of structs](#_yw51rfqnpowf) 6

[Loading of Box2d bodies](#_xzyx81tg059) 6

[Demonstration of this component](#_qj31c62tu1ok) 6

## Description of this Component

The component is used in loading a pre-constructed level for a 2D platformer, using SDL and an option for box2D physics objects, from a single user-supplied .json file into a gameData object(Included with the component). Depending on what methods are called, various objects and vectors are loaded into a GameData. The following guide will describe how to initialise the level loader and how to properly structure your .json file so the LevelLoader can read it.

\*\*\*Important Note\*\*\*

As of the current iteration this .json loader can only handle CircleShape objects and PolygonShape objects that are rectangles from Box2d. This component can not handle irregular polygons.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## How to include this into your project

1. Place the folder named “Levelloader” into your solution folder(Where the “your\_project\_name”.sln file is).
2. Open your project in visual studio and right click your projects solution and open the properties window.
3. Under configuration properties click on “C/C++” menu.
4. In the Additional Include Directories add the line “$(SolutionDir)\Levelloader\include”
5. Open the Linker menu and under Additional Library directories add the line “$(SolutionDir)\Levelloader\lib”
6. In the Input Menu under Linker add the line “LevelLoader.lib” to the Additional Dependencies.

## The important Bits

There are 2 main objects to pay attention to in this component: the LevelLoader object and the GameData object.

LevelLoader

This is the object you initialise in your project. It takes the filepath to wherever your store your .json file in its constructor.

LevelLoader loader(“GameData.json”);

This object is how you get your objects defined in the .json file into your game.

It has 2 main method calls GetLevelData(int level) and LoadBox2dData(b2World\* world, int level).

Notice that both take an int level as an argument. This is how the LevelLoader knows which “level” object in your .json file to open and load in from. Everything that is loaded from the .json file is read into the GameData object. This means you can have as many levels in your game as you please.

GameData

The GameData object included in the component is comprised of static variables. This means that so long as you #include “GameData.h” in your objects, you can access everything that is loaded from the .json file.

## Setting up the .json Loader in your project

### Example: .JSON file creation

This example shows the required setup the .json file needs in order to be read properly by the level loader.

The arrays[] you see below all have an accompanying “Max...” number(i.e. The “Sprite\_List” has a “Max\_Sprites” beside it in the “Sprites” Object). This number tells the loader how many items are in the list. It is important this number is the same as the number of items in the list as if there are more items in the list, not all items will be loaded into the vector and if there are less items than the number states, your program will crash and throw an exception.

Another thing to keep in mind while making your .json is the names of variables in your .json file MUST be the same as the variable described below.

"general\_game\_data": {

"global\_settings": {

"font\_file": "",

"sfx": {

}

},

"main\_menu\_items": {

"menu\_items": [

{

"size": 40,

"text": "Play",

"x": 700,

"y": 300,

"Width": 100,

"Height": 50

}

]

},

“Level1” : {

“background\_image” : “background1.png”,

“max\_game\_objects” : number,

“player\_x” : 500,

“player\_y” : 500,

“GameObjects” : [

{

“X” : 50,

“Y” : 750,

“Width” : 500,

“Height” : 20,

“type” : “Platform”

},

{

“X” : 120,

“Y” : 7000,

“Width” : 10,

“Height” : 50,

“type” : “Wall”

},

….

]

},

"level1Box2D": {

"max\_b2shape": 2,

"b2List": [

{

"x": 100,

"y": 100,

"width": 100,

"height": 100,

"angle": 0,

"density": 10,

"friction": 0.2,

"restitution": 0.2,

"type": "dynamic",

"awake": true,

"shape": "box"

},

{

"x": 300,

"y": 100,

"width": 100,

"height": 100,

"angle": 0,

"density": 10,

"friction": 0.2,

"restitution": 0.2,

"type": "dynamic",

"awake": true,

"shape": "circle",

"radius": 50

}

]

},

### Example: Loading general objects

The basic function of the Loader is the creation of a vector of structs containing an x,y width, height and a string called type. Once the loader has loaded the level into GameData you, the user can use the string type in every struct to initialize your own classes. Also a PlayerX and a playerY is included when getLevelData(int level) is called, set this in the json to create a start position for your player in the level

Here is an example of how to instantiate your objects using variables from the .json

#include “LevelLoader.h”

LevelLoader loader;

// initialize the level loader with a file path to the .json file the levels are located in

loader.Init(“file.json”);

loader.getLevelData(1); //loads the data for the first level into GameDataObject

loader.GameData.GameObjects; //Access the vector holding the structs

std::Vector<\*YourClass> vec; //In the interest of having multiple of one type of object use a vector to store them. Make the vector use pointers so that its size when passing to methods is smaller

for(int i = 0; i < loader.GameData.GameObjects.size(); i++)

{

if(loader..GameObjects.at(i).type == “YourClass)

{

//create your variable here

vec.push\_back(new YourClass(loader..GameObjects.at(i).x,

loader..GameObjects.at(i).y, loader..GameObjects.at(i).width,

loader..GameObjects.at(i).height);

}

}

player.setPosition(loader.GameData.playerX, loader.GameData.Y);

### Loading Box2D Bodies

To load the Box2D bodies you have defined in your .json there is one requirement before calling LoadBox2dData(b2World\* world, int level) and that is to have your b2World object already initialised before calling it. Apart from that it can be called the same as the GetLevelData(int level) from above.

An additional thing to note is once you have your vector of bodies you can use the vector as your for loop for instantiating your Box2D enabled objects.

for (int i = 0; i < GameData::m\_b2BodyVec.size(); i++)

{

Objects.push\_back(new B2Object(r, GameData::m\_b2BodyVec.at(i)));

}

## Features

### Loading of structs

* User calls a method and passes an int.
* Loader finds level + number object in .json.
* Loader attempts to find list of structs and load them into vector in GameData.

### Loading of Box2d bodies

* User calls LoadBox2dData(b2World\* world, int level)
* Loader finds “‘Level’ + number + ‘body’” object in the .json file.
* Loader attempts to instantiate box2D bodies from a list of information found in the object from the .json.
* Places instantiated b2body’s into a vector which is stored in GameData.

## Demonstration of this component

Included with the component is a demo project demonstrating the component. Once you launch the game, press the space bar and you are taken to the demo. Using the 1,2,3 keys(no the number pad keys on the right that some keyboards have) will either reset the currently displayed level or take you to another level with slightly different changes. Keep in mind that every time a key is pressed both methods GetLevelData() and LoadBox2dData() are being called and is actively reading from the .json file.

The demonstration is of a flat plain and 2 box2d objects, one on the left being a polygon shape and the other on the right a circleshape.

**Demo Game Feature**

* On launch game window is a blank screen.
* Pressing the 1-3 keys loads a different level.